Interaction Paradigms for Human-Computer Cooperation in Graphical-Object Modeling

Sandeep Kochhar, Joe Marks, and Mark Friedell

Harvard University

Abstract

Modeling is the creation of graphical objects. It is a tedious task for users to perform, and too complex to be amenable to full automation. The notion of cooperative modeling, where a human and a computer system cooperate to perform the modeling task, therefore, has great appeal. We provide a comparative description of *human-computer cooperative interaction paradigms* for creating graphical objects. This work serves three purposes: 1.) to present a conceptual framework for organizing known paradigms; 2.) to provide a basis for choosing among the set of existing paradigms; and 3.) to expose opportunities for developing new interaction paradigms with certain desirable combinations of characteristics.

Keywords: Graphical user interfaces, modeling, design automation, human-machine interaction, interaction techniques, critics, constraint-based design, cooperative design, automated design of graphical displays.

1 Introduction

Modeling—the process of creating graphical objects—demands the majority of the human effort invested in most computer-graphics applications. In contrast to rendering, the modeling activity is understood poorly and, in general, supported inadequately. The imbalance in the relative developments of modeling and rendering technologies reflects the historically disproportionate emphasis on rendering research at the expense of modeling. This lopsided research focus is changing, however, as we begin to satisfy the challenge of photorealism and as the topic of *visualization* emerges with its demand for rapidly produced graphical depictions of data.

We define *graphical objects* broadly, encompassing, for example, informational graphics (e.g., bar charts and maps), the graphical components of user interfaces (e.g., dialog boxes and windows), and the graphical objects produced with geometric CAD systems (e.g., architectural floor plans and graphical depictions of physical systems). We view the modeling process as a combination of two different activities:

MODELING = DESIGN + ARTICULATION

Design is the more creative and inventive aspect of modeling. It is an exploratory process that produces a conceptual arrangement of the logical elements of an object model, a construct that we refer to as an *object conceptualization.* Articulation is the activity of providing a precise graphical description of an object model, given its conceptualization. For example, when using an architectural CAD system an architect will be faced with a design task when attempting to determine the best collection and organization of rooms in a building; expressing the precise geometry of that floor plan to the computer is an articulation task. Landscape design identifies the locations and types of the topographic features, trees, buildings, and other natural and man-made structures in the scene. The geometric and optical characteristics of these features are provided when the scene is graphically articulated.

Except in a few narrow, well-understood domains, completely automatic object modeling is not possible. In many applications, however, it is possible to bring the power of the computer to bear on the modeling task, with the guidance and cooperation of a human collaborator.[1]

An emerging set of interaction paradigms facilitating human-computer cooperation in modeling has been reported in the recent literature of the computer-graphics, interactive-computing, and artificial-intelligence communities. This paper contributes a comparative descrip-

Contact: Sandeep Kochhar, Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138 (telephone: 617-495-3998, electronic mail: kochhar@endor.harvard.edu.)

This research was supported by contracts and grants from the Lockheed Corporation and U S West Advanced Technologies.

[1]We use the term "user" and "collaborator" interchangeably throughout the paper to refer to the human performing the modeling task.

tion and classification of these paradigms.[2] For the practicing engineer, this provides a basis for choosing the appropriate complement of these paradigms for new applications. For the researcher, this study provides a useful framework for cooperative modeling and indicates unexplored regions of the "organizational space" that may contain significant opportunities for developing new paradigms.

2 Paradigm Organization

To organize the various paradigms conceptually, we consider the extents to which they automate design and articulation (Figure 1). The degree of automaticity in each dimension that the different paradigms offer varies from completely manual to completely automated. This is essentially related to the load distribution between the user and the computer—towards the manual end of the scale, the user is *active* and the system essentially *passive* with respect to the modeling process; the opposite is true towards the fully automated end. Also, in some of the paradigms, the degree of automaticity can be varied dynamically within a particular modeling task.

In addition, the paradigms may be distinguished by other factors:

- nature of the application domain: the kinds of graphical objects that are being modeled and their relation to real-world entities and tasks
- design variability: the range of design variation that can be supported and managed by the paradigm
- exploratory nature of the modeling process: whether the goal is the creation of a single optimal graphical object or the creation of a suite of complementary graphical objects
- user expertise and goals: the degree of user expertise and the user's purpose in creating the graphical object

While our discussion of the individual paradigms considers these factors, we do not use them as major classification characteristics because they do not cover in a meaningful way the complete range of paradigms that we present.

As Figure 1 shows, we classify existing interaction paradigms for graphical-object modeling into six categories, organized with respect to our principal organizational characteristics.[3] These six categories are:

1. Fully manual
2. Constraint-based
3. Critic-based
4. Improver-based
5. Fully automated
6. CCAD (Cooperative CAD)

We discuss below the salient nature of each of the paradigms and their relative strengths and weaknesses. We also present examples of systems based on the paradigms.

3 Interaction Paradigms

3.1 Fully Manual Modeling

The interaction paradigm used in most widely available CAD systems is that of fully manual modeling. These systems primarily support low-level geometric manipulation, with only a few systems offering higher-level design operators. In all cases, the user is responsible for all design decisions. This interaction paradigm is thus characterized by the following properties:

- the user has complete control over the modeling process
- the system is passive with regard to the modeling process

MacDraw and MacPaint are two early examples of fully manual tools for designing 2D graphical objects.[4] GEOMOD [Myer82], AutoCAD [Eyri90] and CADDS[5], all three mechanical CAD systems, and SCHEMA [Norm86], a three-dimensional sketching system for architects, are examples of fully manual tools for designing 3D objects. The AVS [Upso89] and apE [Dyer90] systems support the production of scientific visualizations. Pagemaker[6] and Framemaker[7] are systems for page layout in documents. A number of user-interface toolkits provide support for the manual design of graphical objects for use in user interfaces [Myer89].

[2] Some previous attempts to characterize cooperative human-machine interaction have favored a particular paradigm over others. For example, Fischer *et. al.* [Fisc90] describe the critic-based approach as an exemplar for cooperative interaction. Instead, we have tried to include and classify the full spectrum of interaction paradigms: in our organizational space the critic-based paradigm is only one of several different approaches to human-computer cooperation in modeling.

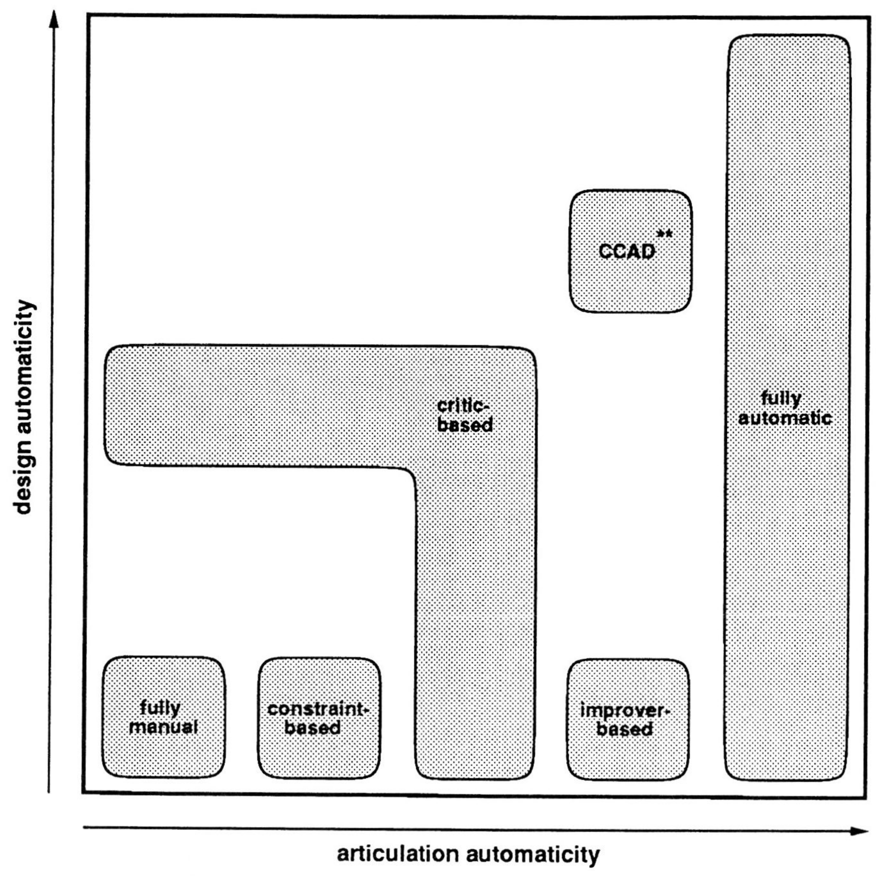
[3] Many of the paradigms can be extended to cover other parts of the organizational space; however, in Figure 1, we show them as occupying that portion of the space that brings out their most salient characteristics.

[4] MacDraw and MacPaint are trademarks of Apple Computer Corporation.

[5] CADDS is a registered trademark of the Computervision division of Prime Computer.

[6] Pagemaker is a trademark of Aldus Corporation.

[7] Framemaker is a trademark of Frame Technology Corporation.



** The automaticity in both design and articulation that are exhibited in a single design session may range widely, hence the proper coordinates of the CCAD paradigm in this figure are problematic, as discussed in Section 3.6.

Figure 1: Conceptual Organization of the Interaction Paradigms

Although all these systems have powerful and effective features, they are fully manual in the sense that the system plays only the role of facilitator for all design and articulation tasks: these tasks are performed essentially by the user.

While the fully manual interaction paradigm has the advantage that the user has complete control over the modeling process, the creation of graphical objects can be a time-consuming and tedious process.

3.2 Constraint-based Modeling

The constraint-based modeling paradigm allows somewhat more automaticity than does the fully manual modeling paradigm. In constraint-based modeling systems, most of the modeling is done manually, except that the system attempts to satisfy a set of constraints as the graphical object evolves. (These constraints can be specified by the user, or may be inherent to the domain.) Thus, as the user manipulates a nascent object, the system may make minor adjustments to ensure that constraints will remain satisfied. Alternatively, the system may restrict the user's options at each step in the modeling process, so that the graphical objects produced do not violate constraints.[8] The main characteristics of this paradigm are:

- the system offers no modeling advice of any kind
- the human user makes all modeling decisions, but his or her options are constrained by the system

[8]The systems cited in the previous subsection might also be considered to constrain the modeling process because they provide a limited repertoire of graphical-subobject types, or because they only allow certain operations to be performed. Our notion of constraint-based modeling is more restrictive, as will become clear in the subsequent discussion.

Although this interaction paradigm may yet prove useful for design tasks, it has so far proven useful only for articulation tasks, as we have indicated in Figure 1.

One of the earliest examples of the constrained-modeling paradigm for articulation is Borning's ThingLab [Born81]. Borning's notion of a constraint is very general: in ThingLab, a constraint consists of a declarative relation over graphical subobjects composed of lines and polygons; a procedure for measuring how well the relation is satisfied; and a set of methods or procedures for satisfying the relation. The user specifies constraints; the ThingLab system then articulates a graphical object that satisfies these constraints. Constraint satisfaction is achieved by applying the user-supplied methods associated with each constraint: this process is guided by a variety of techniques for constraint propagation and constraint relaxation.

Another example of the constrained-modeling paradigm is Nelson's Juno system [Nels85]. For this discussion the relevant aspect of Juno is its method of using geometric constraints to specify locations of two-dimensional points. The primitive graphical subobjects allowed in Juno are lines, arcs, and areas, all of which are defined in terms of points. Juno supports four types of geometric constraint. The first type of constraint is called congruence: two pairs of points, (x, y) and (u, v), are constrained to be congruent by requiring the distance between x and y to equal the distance between u and v. The second constraint type concerns parallelism: two pairs of points, (x, y) and (u, v), are constrained to be parallel by requiring the direction from x to y to parallel the direction from u to v. The third and fourth types of constraint require pairs of points to be aligned horizontally and vertically, respectively. Juno articulates a graphical object that satisfies a given set of constraints using a Newton-Raphson method for constraint satisfaction.

A more recent example of the constrained-modeling paradigm is Sistare's Converge system [Sist91]. Converge differs from ThingLab and Juno in several ways: the graphical subobjects and constraints in Converge are three-dimensional, the constraints are presented to the user as graphical icons that are superimposed on the geometry, and the constraints are satisfied more efficiently through the use of partitioned constraint networks and more aggressive numerical techniques. Figure 2 shows a graphical object produced by Converge in which various constraints are used to control the form of the geometry. The legend explains the meaning of the various constraint icons used in the figure.

The examples discussed above all concern the articulation task: the user is responsible for deciding which graphical subobjects to include and for stating the relations that should hold between them; the system is responsible for instantiating a graphical object that comprises the user-specified subobjects and that satisfies the user-specified relations or constraints. Many articulation tasks are best thought of as constraint-satisfaction tasks, so it is perhaps not surprising that the constraint-based paradigm has been most successful in this area. While it is certainly possible to use the concept of constrained modeling for design, we know of no system that takes this approach.

Constraint-based modeling systems offer the advantage that the final graphical objects are guaranteed to be "valid" in some sense. Sometimes, however, the interactions between subobjects that are subject to constraints can be difficult for the user to anticipate fully, and the modeling task can still be rather tedious.

3.3 Critic-based Modeling

Next along the continuum of increasing automaticity is the paradigm of modeling using critics. Critics are user-invoked agents that respond to user-generated graphical objects by providing *criticisms.* The range of criticisms offered can be broad, from notification of low-level geometric constraint violations to high-level critiques of object conceptualizations. The main characteristics of this paradigm are:

- the user is still required to perform the entire modeling task manually
- critics identify flaws in design and articulation, but remedies must be applied by the human collaborator

Thus, critics do not autonomously develop graphical objects, but detect suboptimal aspects of the emerging objects being created by the human user. They provide feedback to the user and enable him or her to develop a better object. Lemke [Lemk90a] argues for the necessity of having critics in any cooperative problem-solving system. A detailed discussion and survey of the critic-based approach is provided by Fischer *et. al.* [Fisc90].

An example of a design system based on critiquing is CRACK [Fisc88, Fisc89]. CRACK's knowledge-based critiquing component encodes design principles about assembling kitchen appliances into functional kitchen layouts. These rules are based on building codes, safety standards and functional preferences. The user is expected to resolve criticisms based on building codes and safety standards, except in exceptional cases; those based on functional preferences can be viewed as optional suggestions. CRACK can also provide default explanations (consisting of "canned" text) for its criticisms if the user so requests.

As another example, Oxman and Gero [Oxma87] present PREDIKT as an expert system that can be used for both "design diagnosis" (critiquing) and "design synthesis" (automatic articulation). PREDIKT carries out both these tasks in the preliminary stages of the design of domestic kitchens. In the "diagnosis" mode, the system criticizes (e.g., "proportions are inadequate") and

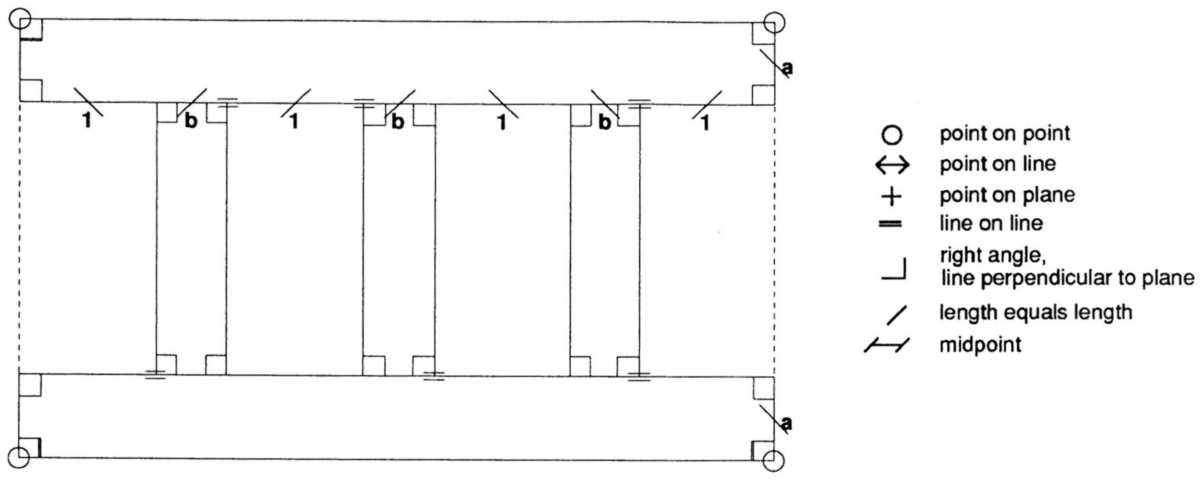


Figure 2: A Graphical Object Produced by the Converge System

evaluates designs (e.g., "light is sufficient"), based on encoded knowledge relating to requirements for kitchens. The comments offered by the system as it evaluates a design can be viewed as approval of the steps taken so far by the human designer. An interesting feature of PREDIKT is that the same knowledge-base is used in the critiquing and automatic articulation steps.

Other examples of critic-based systems that help with articulation tasks include mechanical and electrical CAD systems that can perform structural-integrity tests, interference checks, and layout-feasibility tests.

The critics described so far have to be explicitly invoked by the user. It is thus possible that critics might be invoked too late in the modeling process, after a major incorrect decision had already been made. An alternative is to have *active* critics [Fisc90] (sometimes called *daemons*). Active critics "watch over" the user's actions and warn the user asynchronously (that is, without waiting for user invocation) when critical information needs to be communicated or when flaws are detected.

FRAMER [Lemk90a, Lemk90b, Fisc90] is an example of an active-critic-based system for designing user interfaces. FRAMER contains a knowledge base of design rules for program frameworks that evaluate the completeness and syntactic correctness of the user interface being designed. The active critics used in FRAMER are partitioned into mandatory and optional ones, similar to the critics in CRACK. Messages from these critics are continuously presented to the user in the form of a checklist that the user cannot permanently ignore. Figure 3 shows an example from FRAMER.[9] The checklist produced by the critics is shown in the window towards the center, titled "Things to take care of."

Critics are obviously useful for detecting and pointing out suboptimal aspects in an emerging graphical object. The major challenge is to be able to capture design knowledge, both for design and articulation, in the form of predicates that can be tested. The role of a critic can be enhanced if the system is also able to offer rationales for its criticisms [Fisc89, Fisc90].

[9]This figure is reproduced from [Fisc90] with permission.

3.4 Improver-based Modeling

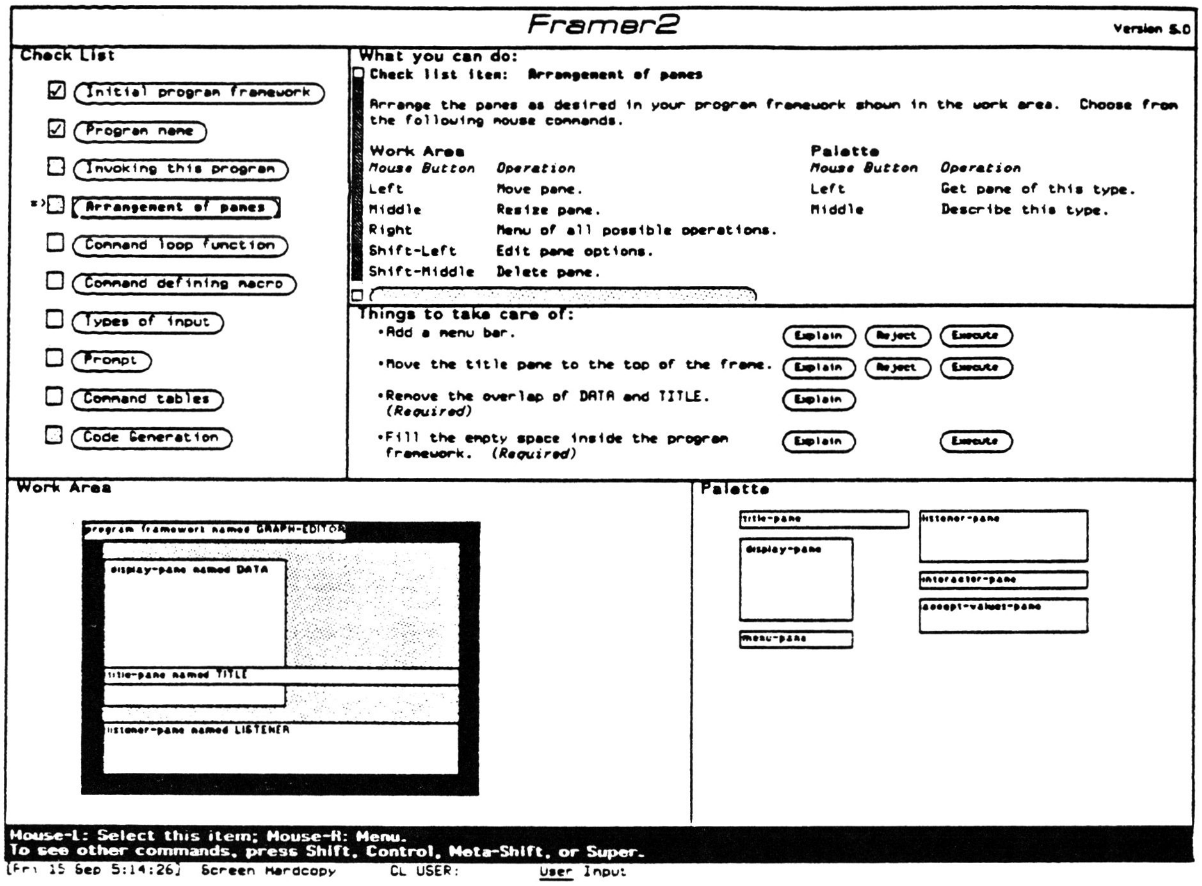
An improver is a highly automatic agent that modifies ("perturbs") a completed graphical object produced by the user in order to improve it. As indicated in Figure 1, however, the improver-based paradigm has been applied only to articulation tasks. The essential difference between the improver-based paradigm and the critic-based paradigm is that the latter need only identify shortcomings (and maybe offer abstract suggestions), whereas the former must also attempt to rectify them. The main characteristics of improvers are:

- only a completed graphical object can be improved
- flaws in the object are identified *and* remedied automatically
- the human collaborator may not be the originator of all aspects of an improved graphical object

The essence of the improver-based paradigm is illustrated in Figures 4 and 5.[10] The graphical object—a network diagram—in Figure 4 is obviously conceptually complete, but the articulation of the conceptualization is lacking. Figure 5 shows the result of improving the articulation, a process Pavlidis and Van Wyk call "beautification."

The improver-based algorithm in [Pavl85] has two components. The first component infers the relations

[10]These figures are reproduced from [Pavl85] with permission.



Check List

- [✓] Initial program framework
- [✓] Program name
- [ ] Invoking this program
- [ ] Arrangement of panes
- [ ] Command loop function
- [ ] Command defining macro
- [ ] Types of input
- [ ] Prompt
- [ ] Command tables
- [ ] Code Generation

What you can do:

Check list item: Arrangement of panes

Arrange the panes as desired in your program framework shown in the work area. Choose from the following mouse commands.

Work Area

Mouse Button	*Operation*
Left	Move pane.
Middle	Resize pane.
Right	Menu of all possible operations.
Shift-Left	Edit pane options.
Shift-Middle	Delete pane.

Palette

Mouse Button	*Operation*
Left	Get pane of this type.
Middle	Describe this type.

Things to take care of:

- Add a menu bar.
- Move the title pane to the top of the frame.
- Remove the overlap of DATA and TITLE. *(Required)*
- Fill the empty space inside the program framework. *(Required)*

Work Area

program framework named GRAPH-EDITOR
display-pane named DATA
title-pane named TITLE
listener-pane named LISTENER

Palette

title-pane
listener-pane
display-pane
interactor-pane
accept-values-pane
menu-pane

Mouse-L: Select this item; Mouse-R: Menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.

[Fri 15 Sep 5:14:26] Screen Hardcopy CL USER: User Input

Figure 3: Example of Active Critics in FRAMER (from [Fisc90])

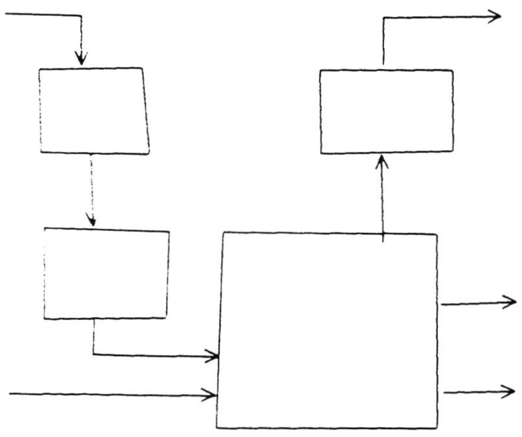


Figure 4: Network Diagram Before Improvement (from [Pavl85])

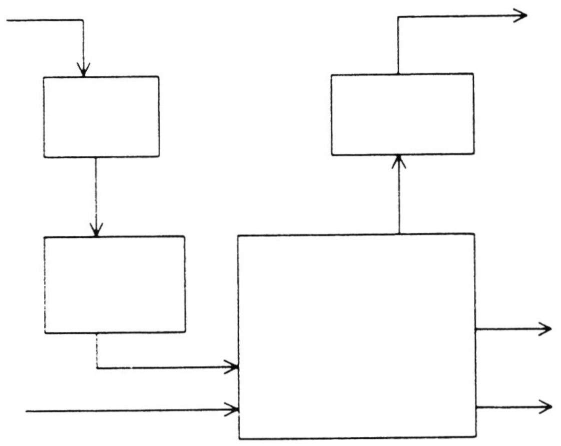


Figure 5: Network Diagram After Improvement (from [Pavl85])

or constraints that should hold between graphical subobjects (composed of line segments and polygons) in a given image. In other words, the system first tries to infer the relevant aspects of an object conceptualization from a completed graphical object. (This inference step is obviously necessary if only an image is available, but for some applications a representation of the object conceptualization might be available, rendering the inference step moot.) The constraints considered are length and slope congruence between line segments, line-segment collinearity, horizontal and vertical alignment of points, and various "negative" constraints to ensure that subobjects do not overlap or otherwise interact in a deleterious way. Given a set of constraints, the system modifies the arrangement of graphical subobjects to satisfy the constraints. In this respect, the beautifier system of Pavlidis and Van Wyk is reminiscent of the constraint-based systems considered earlier.

Another example of the improver-based paradigm is Weitzman's DESIGNER system [Weit86], an interactive tool for creating graphical interfaces to instructional systems. DESIGNER has three component processes: an analysis process that infers aspects of an object conceptualization from a graphical object; a critiquing process that identifies elements of the object conceptualization that have not been articulated satisfactorily; and a synthesis process that suggests methods for modifying the articulation of the conceptualization. Some unique aspects of DESIGNER are the primitive graphical subobjects it uses (icons with properties of color, size, type, and shape), the subobject relations it recognizes and can attempt to modify (perceptual organization by similarity, proximity, and repetition), and its ability to accommodate different graphical styles.

The suitability of the improver-based paradigm for better articulation is fairly obvious. The need to infer indirectly aspects of an object conceptualization from a completed graphical object is a problem that future improver-based systems may be able to avoid. The application of the improver-based paradigm to the design task is likely to prove quite hard, because it would appear to be necessary to know a great deal about the application domain, and the purpose of a graphical object (user objectives) in order to improve its design.

3.5 Fully Automated Modeling

At the far end of the automaticity spectrum lies fully automated modeling. In this paradigm, the system is completely responsible for design and/or articulation. The main characteristics of this approach are:

- the system is completely responsible for one or both aspects of the modeling task
- the user is passive with respect to one or both aspects of the modeling task

Full modeling automation has been achieved in several domains, but only for a small number of well-defined applications:

- *Iconic Displays.* The VIEW system [Frie82, Frie84] generates icon-based displays that depict answers to database queries. The displays are tailored to the user's task, identity, and nature of the query.
- *Chart Graphics.* Mackinlay's APT system [Mack86] automates the design and articulation of chart graphics that communicate arbitrary relational information. Roth and Mattis [Roth90] have extended Mackinlay's approach to allow for greater design variation by enriching the characterization of the input data; their system is part of a multimedia explanation generator for a financial-analysis application [Roth89].
- *Three-Dimensional Illustrations.* The APEX [Fein85] and IBIS [Seli89] systems produce illustrations that depict objects and actions in the physical world. The illustrations satisfy communicative goals generated automatically by a multimedia explanation generator [Fein90].
- *User-Interface Displays.* Several systems have been built that are capable of generating automatically the graphical objects needed for user-interface displays [Aren88, Kim90, Wiec90].
- *Network Diagrams.* The ANDD system [Mark90a, Mark90b] designs and articulates network diagrams to communicate information represented in arbitrary attributed graphs. This system will provide part of a multimedia explanation capability for a collaborative-planning system [Gros90].

These systems share some common characteristics: they all communicate very specific kinds of information represented in specialized formats; their primary design task is to map symbolic information onto an expressive and effective graphical depiction; and they target applications where human collaboration is unnecessary (because of limited design variation) or impossible (because of the application context). These systems also differ significantly in many respects: the graphical conventions and styles that govern the different kinds of display vary greatly (e.g., the issue of 3D viewing parameters only arises in the APEX and IBIS systems, and layout-related perceptual organization is of primary concern to only the ANDD system), causing great variation in the basic algorithmic paradigms used by the different systems.

The systems mentioned above automate both design and articulation tasks completely. The fully automated approach has also been applied to just articulation, especially to those articulation tasks involving complex or tedious layout problems that require combinatorial search:

- *Floor-Plan Layout.* Several researchers in architectural CAD have proposed schemes for automating the articulation of conceptualizations of floor plans. Earlier approaches, such as those of Mitchell [Mitc76], Bloch [Bloc79], Galle [Gall81], Steadman [Stea83], and Rinsma [Rins88], were based on breaking down the task into two stages: topological layout and geometric (dimension) assignment. More recently, researchers have focused on approaches that use generative rules with restrictive applicability predicates similar to those used in expert systems. Examples of these can be found in Oxman and Gero's PREDIKT [Oxma87] system (described in section 3.3) for producing domestic kitchen layouts, and Flemming's LOOS system [Flem86, Flem89] for creating floor plans that incorporate a wide range of design considerations.

- *Network-Diagram Layout.* No articulation task has received more attention than network-diagram (or graph) layout. [Eade89] lists more than 180 references in an extensive annotated bibliography devoted to algorithms for this task. Almost all network-diagram-layout research has concentrated on the issue of readability, as judged in terms of aesthetic criteria such as the minimization of edge crossings.[11] Different algorithms have been proposed to take advantage of different types of graph (tree-like, planar, directed, and undirected) and to generate layouts according to various graphical standards (e.g., layouts that have straight-line, polyline, or orthogonal edges).

- *Cartographic Layout. Contour drawing* [Yoel84, Sabi85, Dobk90], *label placement* [Yoel72, Hirs82, Ahn84, Zora86, Jone89, Roes89], and *line generalization* [McMa87][12] are three articulation tasks that arise in the design of maps and that have been automated fully with varying degrees of success.

- *Page Layout.* Feiner describes a fully automated approach to page layout in [Fein88].

At this point, fully automated modeling is an attractive option only in a limited number of situations, namely those where human collaboration is impossible (e.g, time critical applications), where design variation is very limited, or where the articulation task involves tedious or complex combinatorial search. In most other situations, a human collaborator can play a useful and active role in the modeling process. Nevertheless, research on fully automated modeling serves a useful purpose outside its narrow domain of direct applicability by providing ideas and algorithms that can be incorporated into more cooperative modeling paradigms.

[11]For a different approach to network-diagram layout that concentrates on perceptual organization, see [Mark90b].

[12]Geographic features are rendered in less detail after a reduction in scale: this simplification process is called generalization.

3.6 Cooperative Computer-Aided Design

Cooperative computer-aided design (CCAD)[13] [Koch90a, Koch90b, Koch90c] is a paradigm for combining the strengths of the human user and the computer by interspersing guiding design operations by the system user with partial exploration of design alternatives by the computer. While the salient impression of a CCAD system is conveyed in Figure 1, the automaticity in both design and articulation that are exhibited in a single design session may range widely. As a result, the proper coordinates of the CCAD paradigm in Figure 1 are problematic, and this makes CCAD somewhat distinct from the other paradigms reviewed in this paper.

In the CCAD paradigm, the user expresses initial design decisions in the form of a partial design and a set of properties that the final design must have. The user then initiates the generation by the system of alternative *partial* developments of the initial design subject to a "language" (grammar) of valid designs. The results are then structured in a spatial framework through which the user moves ("browses") to explore the alternatives. The user selects the most promising partial design, refines it manually, and then requests further automatic development. This process continues until a satisfactory design is completed. CCAD also provides mechanisms for user control over the automated generation process. These mechanisms serve to specify constraints on designs, restrict the activation of design rules, and focus development on specific parts of designs.

Thus, in the CCAD paradigm the degree of automation offered at any step in the design process can vary from fully manual to fully automated: the user can ignore design alternatives produced by the system (resulting in fully manual design), can allow the system to choose the best alternative (resulting in a higher degree of automaticity), or can choose from the system-generated designs (in which case the design task is essentially shared). The latter case is the most interesting: the human collaborator is guiding the design by making critical design choices, and the system is performing the relatively low-level aspects of the design task. The main characteristics of CCAD are:

- the design task is shared between user and system (both the user and the system are *active*) to varying degrees

- the system generates one or more design alternatives at various stages in the design process

- the system provides a "browsing" capability to aid the user in choosing between system-generated design alternatives

[13]While CCAD can be used to support both the design and articulation tasks (the two tasks are essentially merged), its name reflects the fact that this paradigm was developed to support the design task primarily.

We give examples of the above mechanisms of CCAD from FLATS (Figure 6) [Koch90a, Koch90b, Koch90c]—a prototype CCAD system for the design of small architectural floor plans. In the figure, the nascent design—a floor plan with an entry, an external region and an internal region—is shown on the manual modeling system (the window is in the top left of the figure). The two windows titled "Rewriting Rule Visualization Interface" and "Constraints on Derived Attributes" allow the user to control how far the system develops the nascent design. The former window graphically presents to the user the rules in the underlying staged (programmed) generative grammar and allows the user to restrict the automated generator to use only certain rules in the current design cycle; in the example, the user has restricted the generator to use only the highlighted rules to subdivide the external region labeled *ext* in the nascent design. The latter window lists constraints that the user specified: (1) one to five rooms, (2) at most one bedroom, one kitchen, one dining room, and one living room, and (3) at most two bathrooms. The two windows towards the right, titled "World View" and "Current Data Surface," compose the Browsing System, which allows the user to graphically explore the design alternatives. The World View shows the entire data surface in miniaturized form, along with highlights showing the current data surface. Finally, the Current Data Surface shows the design alternatives (numbering 85) that satisfy the user-specified criteria. The scrollbars can be used to examine different portions of the data surface. The user can pick any of these alternatives, transfer it to the modeling system, and repeat the above process.

Other examples of systems based on the CCAD paradigm include Friedell and Schulmann's Landscape Generator [Frie90], Jakiela's "suggestion-making" interfaces [Jaki90], Todd's Mutator system [Hagg91], and the IVE design system [Koch91]. The Landscape Generator uses an underlying generative mechanism to model architectural landscapes subject to user-specified constraints and features that the final landscapes must incorporate. Jakiela describes a suggestion-making mechanical CAD system that provides a cooperative approach for mechanical modeling: the system can "suggest" improvements at every modeling step (or at the end of the modeling process). His system can be viewed as a restricted CCAD system, because the system cannot autonomously model an object, and because modeling proceeds in small steps. The Mutator system is a modified solid-modeling system in which forms are composed of geometric primitives (e.g., spheres, cubes, and cylinders) that can be altered by shape-distorting operations (e.g., twisting, stretching, and uniting). The system can generate variations on a given form by applying these operations randomly. The user then selects from among the randomly generated alternatives the form that is to be evolved further. Mutator has been used by artist William Latham to create several spectacular animations. The CCAD component of IVE (Integrated Visualization Environment) is used for the design scientific visualizations by the combination of primitive graphical "features," in accordance with a set of design rules. The system can automatically present the user with novel visualizations, which the user can then refine to suit his or her requirements

The CCAD paradigm is most useful in applications with a high degree of design variability that require the user to explore many design alternatives.

4 Conclusions

The foregoing survey of cooperative interaction paradigms for modeling graphical objects shows that each has its own very different characteristics, strengths, and weaknesses. Typically a paradigm determines a style of interaction for both design and articulation, and the paradigms can be ordered with respect to automaticity in these two dimensions.

Our analysis suggests some new directions for future work, indicated by the unexplored regions of Figure 1. Constraint-based modeling and improver-based modeling might be usefully extended to cover design. Fully automatic design, with manual articulation, might be useful for some modeling tasks that require complete user control of the articulation process. Furthermore, the concepts and algorithms developed for fully automated modeling might be used to expand the capabilities of all the other paradigms in the spirit of CCAD.

Acknowledgments

We would like to thank the members of the Computer Graphics Group at Harvard for many useful and interesting discussions. We are also grateful to the anonymous reviewers for their useful suggestions, particularly with regard to the organization of Figure 1.

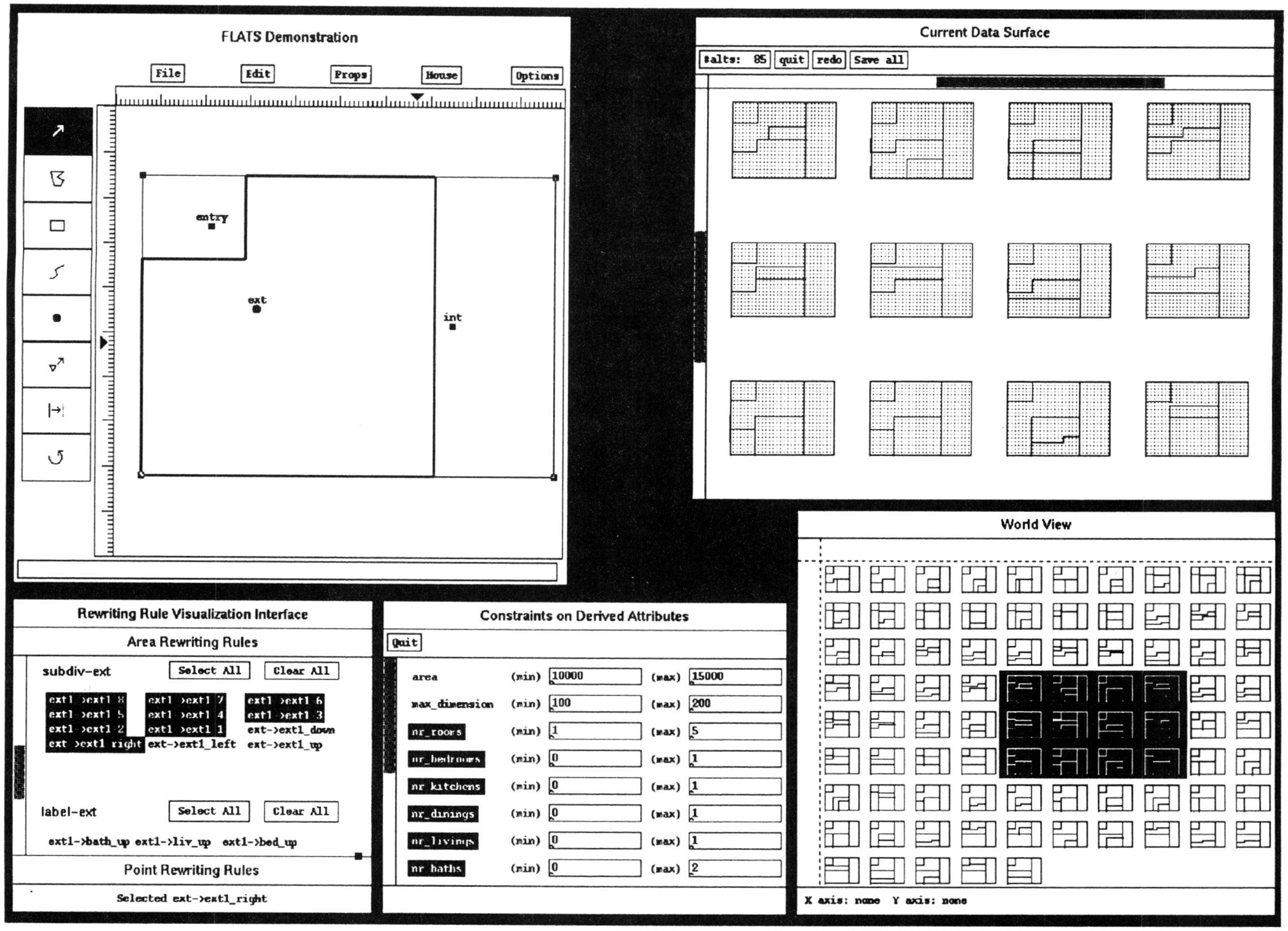
References

[Ahn84] Ahn, J. and Freeman, H. 1984. "A program for automatic name placement," *Cartographica*, **21(2&3)**:101–109, originally published in *Proceedings of the Sixth International Symposium on Automated Cartography (Auto-Carto Six)*, Ottawa/Hull, October 1983.

[Aren88] Arens, Y., Miller, L., Shapiro, S., and Sondheimer, N. 1988. "Automatic construction of user-interface displays," *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI '88)*, pages 808–813.

[Bloc79] Block, C. 1979. "Catalogue of small rectangular plans," *Environment and Planning B*, **6**:155–190.

[Born81] Borning, A. 1981. "The programming language aspects of ThingLab, a constraint-oriented simulation laboratory," *ACM Transactions on Programming Languages and Systems*, **3(4)**:353–387, October.



subdiv-ext

ext1->ext1 8 ext1->ext1 7 ext1->ext1 6
ext1->ext1 5 ext1->ext1 4 ext1->ext1 3
ext1->ext1 2 ext1->ext1 1 ext->ext1_down
ext->ext1 right ext->ext1_left ext->ext1_up

label-ext

ext1->bath_up ext1->liv_up ext1->bed_up

Selected ext->ext1_right

area	(min)	10000	(max)	15000
max_dimension	(min)	100	(max)	200
nr_rooms	(min)	1	(max)	5
nr_bedrooms	(min)	0	(max)	1
nr_kitchens	(min)	0	(max)	1
nr_dinings	(min)	0	(max)	1
nr_livings	(min)	0	(max)	1
nr_baths	(min)	0	(max)	2

Figure 6: An Example of FLATS in Use

[Dobk90] Dobkin, D. P., Levy, S. V. F., Thurston, W. P., and Wilks, A. R. 1990. "Contour tracing by piecewise linear approximation," *ACM Transactions on Graphics*, **9(4)**:389–423, October.

[Dyer90] Dyer, D. S. 1990. "A dataflow toolkit for visualization," *IEEE Computer Graphics and Applications*, **10(4)**:60–69.

[Eade89] Eades, P. and Tamassia, R. 1989. Algorithms for drawing graphs: An annotated bibliography. Brown University Dept. of Computer Science Technical Report No. CS-89-09. October, Revised Version.

[Eyri90] Eyrich, G. and Wallach, P. 1990. *Drafting with AutoCAD*. Mithcell Publishing Inc., Watsonville, CA.

[Fein85] Feiner, S. 1985. "APEX: An experiment in the automated creation of pictorial explanations," *IEEE Computer Graphics and Applications*, November.

[Fein88] Feiner, S. 1988. "A grid-based approach to automating display layout," *Proceedings of Graphics Interface '88*, pages 192–197, June 6-10, Edmonton, Canada.

[Fein90] Feiner, S. and McKeown, K. 1990. "Coordinating text and graphics in explanation generation," *Proceedings of AAAI '90*, pages 442–449, August, Boston, Massachusetts.

[Fisc88] Fischer, G. and Morch, A. 1988. "CRACK: A critiquing approach to cooperative kitchen design," *Proceedings of the ACM International Conference on Intelligent Tutoring Systems*, pages 176–185, May.

[Fisc89] Fischer, G., McCall, R., and Morch, A. 1989. "Design environments for constructive and argumentative design," *Proceedings of CHI '89*, pages 269–275, May.

[Fisc90] Fischer, G., Lemke, A., Mastaglio, T., and Morch, A. 1990. "Using critics to empower users," *Proceedings of CHI '90*, pages 337–347, April.

[Flem86] Flemming, U. 1986. "On the representation and generation of loosely packed arrangements of rectangles," *Environment and Planning B: Planning and Design*, **13**:189–205.

[Flem89] Flemming, U., Coyne, R. F., Glavin, T., Hsi, H., and Rychener, M. D. 1989. "A generative expert system for the design of building layouts," *Final report*, Engineering Design Research Center, Carnegie-Mellon University.

[Frie82] Friedell, M., Barnett, J., and Kramlich, D. 1982. "Context-sensitive graphic presentation of information," *Computer Graphics*, **16(3)**:181–188.

[Frie84] Friedell, M. 1984. "Automatic synthesis of graphical object descriptions," *Computer Graphics*, **18(3)**:53–62.

[Frie90] Friedell, M. and Schulmann, J. 1990. "Constrained, grammar-directed generation of landscapes," *Proceedings of Graphics Interface '90*, pages 244–251, May 14-18, Halifax, Canada.

[Gall81] Galle, P. 1981. "An algorithm for the exhaustive generation of building floor plans," *Communications of the ACM*, **24**:813–825.

[Gros90] Grosz, B. and Sidner, C. 1990. "Plans for discourse," in P. Cohen, J. Morgan, and M. Pollack (Eds.), *Intentions in Communication*. Cambridge: Bradford Books, MIT Press.

[Hagg91] Haggerty, M. 1991. "About the cover: Evolution by aesthetics," *IEEE Computer Graphics and Applications*, **11(2)**:5–9.

[Hirs82] Hirsch, S. A. 1982. "An algorithm for automatic name placement around point data," *The American Cartographer*, **9(1)**:5–17.

[Jaki90] Jakiela, M. 1990. "Augmenting mechanical design with suggestion-making interfaces," *Working Notes, AAAI Spring Symposium Series*, pages 60–64, March.

[Jone89] Jones, C. 1989. "Cartographic name placement with prolog," *IEEE Computer Graphics and Applications*, **9(5)**:36–47, September.

[Kim90] Kim, W. C. and Foley, J. D. 1990. "DON: User interface presentation design assistant," *1990 SIGGRAPH Symposium on User Interface Software and Technology (UIST '90)*, pages 10–20, October 3-5, Snowbird, Utah.

[Koch90a] Kochhar, S. 1990. *Cooperative Computer-Aided Design*. PhD dissertation, Harvard University.

[Koch90b] Kochhar, S. 1990. "A prototype system for design automation via the browsing paradigm," *Proceedings of Graphics Interface '90*, pages 156–166, May 14-18, Halifax, Canada.

[Koch90c] Kochhar, S. and Friedell, M. 1990. "User control in cooperative computer-aided design," *1990 SIGGRAPH Symposium on User Interface Software and Technology (UIST '90)*, October 3-5, Snowbird, Utah.

[Koch91] Kochhar, S., Friedell, M., and LaPolla, M. 1991. "Cooperative, computer-aided design of scientific visualizations," *submitted*.

[Lemk90a] Lemke, A. 1990. "Cooperative problem solving systems must have critics," *Working Notes, AAAI Spring Symposium Series*, pages 73–75, March.

[Lemk90b] Lemke, A. and Fischer, G. 1990. "A cooperative problem solving system for user interface design," *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI '90)*, pages 479–484, August.

[Mack86] Mackinlay, J. 1986. "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, **5(2)**.

[Mark90a] Marks, J. 1990. "A syntax and semantics for network diagrams," *Proceedings of the IEEE 1990 Workshop on Visual Languages*, pages 104–110, October, Skokie, Illinois.

[Mark90b] Marks, J. and Reiter, E. 1990. "Avoiding unwanted conversational implicatures in text and graphics," *Proceedings of AAAI '90*, August, Boston, Massachusetts.

[McMa87] McMaster, R. B. 1987. "Automated line generalization," *Cartographica*, **24(2)**:74–111, Summer.

[Mitc76] Mitchell, W., Steadman, J., and Liggett, R. 1976. "Synthesis and optimization of small rectangular floor plans," *Environment and Planning B: Planning and Design*, **3(1)**:37–70.

[Myer82] Myers, W. 1982. "An industrial perspective on solid modeling," *IEEE Computer Graphics and Applications*, **2(2)**:86–97.

[Myer89] Myers, B. 1989. "User-interface tools: Introduction and survey," *IEEE Software*, pages 15–24, January.

[Nels85] Nelson, G. 1985. "Juno, a constraint-based graphics system," *Computer Graphics*, **19(3)**:235–243.

[Norm86] Norman, M. and Dillon, K. 1986. Draft SCHEMA user's manual. T.R. no. LCGSA-86-x (draft), Graduate School of Design, Harvard University.

[Oxma87] Oxman, R. and Gero, J. S. 1987. "Using an expert system for design diagnosis and design synthesis," *Expert Systems*, **4(1)**:4–15.

[Pavl85] Pavlidis, T. and Van Wyk, C. 1985. "An automatic beautifier for drawings and illustrations," *Computer Graphics (Proceedings of SIGGRAPH '85)*, **19(3)**, July.

[Rins88] Rinsma, I. 1988. "Rectangular and orthogonal floorplans with required room areas and tree adjacency," *Environment and Planning B: Planning and Design*, **15**:111–118.

[Roes89] van Roessel, J. W. 1989. "An algorithm for locating candidate labeling boxes within a polygon," *The American Cartographer*, **16(3)**:201–209.

[Roth89] Roth, S., Mattis, J., and Mesnard, X. 1989. "Graphics and natural language as components of automatic explanation," in J. Sullivan, and S. Tyler (Eds.), *Architectures for Intelligent Interfaces: Elements and Prototypes*, Reading, MA: Addison-Wesley.

[Roth90] Roth, S. and Mattis, J. 1990. "Data characterization for intelligent graphics presentation," *Proceedings of CHI '90*, pages 193–200, April.

[Sabi85] Sabin, M. A. 1985. "Contouring — The state of the art," in R.A.Earnshaw (Ed.), *Fundamental Algorithms for Computer Graphics*, pages 99–108, New York: Springer-Verlag.

[Seli89] Seligmann, D. and Feiner, S. 1989. "Specifying composite illustrations with communicative goals," *ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '89)*, pages 1-9, November 13-15, Willamsburg, VA.

[Sist91] Sistare, S. 1991. "Graphical interaction techniques in constraint-based geometric modeling," *Proceedings of Graphics Interface '91*, June 3-7, Calgary, Canada.

[Stea83] Steadman, J. 1983. *Architectural Morphology*. Pion, London.

[Upso89] Upson, C., Faulhaber, T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., and van Dam, A. 1989. "The application visualization system: A computational environment for scientific visualization," *IEEE Computer Graphics and Applications*, **9(4)**:30–42.

[Weit86] Weitzman, L. 1986. Designer: A knowledge-based graphic design assistant. ICS Report 8609, University of California, San Diego. July.

[Wiec90] Wiecha, C., Bennett, W., Boies, S., Gould, J., and Greene, S. 1990. "ITS: A tool for rapidly developing interactive applications," *ACM Transactions on Information Systems*, **8(3)**:204–236, July.

[Yoel72] Yoeli, P. 1972. "The logic of automated map lettering," *The Cartographic Journal*, **9(2)**:99–108, December.

[Yoel84] Yoeli, P. 1984. "Cartographic contouring with computer and plotter," *The American Cartograher*, **11(2)**:139–155.

[Zora86] Zoraster, S. 1986. "Integer programming applied to the map label placement problem," *Cartographica*, **23(3)**:16–27.